2010 Fourth ACM/IEEE International Symposium on Networks-on-Chip

Comparing Energy and Latency of Asynchronous and Synchronous NoCs for Embedded SoCs

Daniel Gebhardt
School of Computing
University of Utah
gebhardt@cs.utah.edu

Junbok You Kenneth S. Stevens
Dept. of Electrical and Computer Engineering
University of Utah
{jyou , kstevens}@ece.utah.edu

***Abstract*—Power consumption of on-chip interconnects is a primary concern for many embedded system-on-chip (SoC) applications. In this paper, we compare energy and performance characteristics of asynchronous (clockless) and synchronous network-on-chip implementations, optimized for a number of SoC designs. We adapted the COSI-2.0 framework with ORION 2.0 router and wire models for synchronous network generation. Our own tool, ANetGen, specifies the asynchronous network by determining the topology with simulated-annealing and router locations with force-directed placement. It uses energy and delay models from our 65 nm bundled-data router design. SystemC simulations varied traffic burstiness using the self-similar b-model. Results show that the asynchronous network provided lower median and maximum message latency, especially under bursty traffic, and used far less router energy with a slight overhead for the inter-router wires.**

I. Introduction

Embedded, energy-constrained SoC designs can be roughly separated into two classes: platform-based and fixed-function (also called application-specific). The former is concerned with being able to perform a wide variety of tasks, many of which cannot be foreseen at design time. The latter is targeted towards a particular function, or a few functions, that have known properties. A fixed-function design might consist of a number of highly specialized cores and memories, and fewer general-purpose processors. The network-on-chip (NoC) of both these classes should be optimized for minimal energy usage while meeting the predicted performance requirements; however, the application-specific NoC may be more specialized as it has *a priori* knowledge of the communication patterns between cores. This is in contrast to general-purpose interconnects that are often evaluated with traffic patterns such as spatially-uniform, bit-transpose, etc.. The domain of this work is the fixed-function, rather than the platform-based SoC.

Some globally-asynchronous locally-synchronous (GALS) interconnect solutions rely on a clock, either with standard synchronous clock distribution, or a mesochronous method. However, an asynchronous (also called clockless) network has a number of potential advantages over a clocked network in a GALS environment. Standard arguments for asynchronous circuit design include robustness to process/voltage/temperature variation, average-case instead of worst-case performance, and other such points. However, there are also many NoC-specific arguments. In a synchronous NoC, the clock tree for all routers and pipeline buffers can consume significant power as shown in a heterogeneous network [1], and in a large CMP (chip multiprocessor) 33% of router power [2]. Many SoC designs have quite bursty and "reactive" traffic. In this case, asynchronous methods are beneficial in that they consume little dynamic power during periods of low traffic without relying on clock gating techniques.

Available bandwidth on each asynchronous link can be independently set, to some extent, by wirelength between routers, link pipeline depth, or by varying the physical wire properties (metal layer, width, and spacing). This is potentially useful when bandwidth requirements on core-to-core paths vary considerably. This is in contrast to clocked networks which commonly use a single frequency for all routers and is wasteful to those paths not requiring high bandwidth. A clocked NoC can use discrete "islands" of differing clock speeds to achieve a similar effect, but in a much coarser-grained fashion.

Design automation techniques are commonly used to generate a NoC for a specific SoC design. These methods can decrease time of development in commercial products or allow a researcher to explore a larger design space. The NoC solution is chosen based on some metric, usually a function of energy and performance. In the optimization process, potential solutions must be evaluated for quality, and this often requires an abstracted model of the SoC characteristics.

This abstraction can be done at a variety of levels depending upon completeness or availability of the SoC design and NoC components. Ideally, one could simulate the exact functionality of the various cores composing the design, and the NoC would be fully implemented to model the communication. Unfortunately, this method is labor and simulation-time intensive, and not a good choice for early-exploration of the NoC design space. As usual, tradeoffs must be made as function becomes more abstracted.

A commonly used abstraction used in the literature has been titled a *communication trace graph* [3] (CTG) or a *core graph*. A *path* describes pairs of source and destination cores, and the particular links and routers a packet traverses. The CTG has a n-tuple of values per path, but often includes average expected traffic rate per path and sometimes a latency requirement of a packet. An example CTG is shown in Figure 1 that we use in our evaluation.

To our knowledge, there does not exist published methods to aid in automating high-level asynchronous NoC optimization

978-0-7695-4053-5/10 $26.00 © 2010 IEEE
DOI 10.1109/NOCS.2010.21

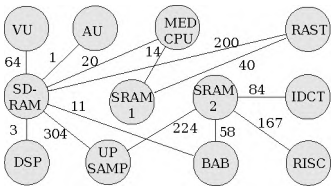


Figure 1: Example CTG graph. Edge weights are in MBytes/s.

for fixed-function SoCs. This is in contrast to techniques that utilize synchronous tools for implementing a specific network [4]. In this work, we give an overview of our circuits and design automation techniques, and compare the resulting asynchronous NoC to a synchronous one generated by an existing tool. We also show that adding a measure of bursty traffic to a CTG design abstraction leads to a more conclusive NoC evaluation. Also unique is our SystemC simulator that models asynchronous routers, and importantly, the link delay as a function of wire length between routers.

This paper is organized as follows. Section II gives an overview of related work. The synchronous network generation framework is discussed in Section III. Our asynchronous router is discussed in Section IV at a circuit level. Section V describes our methodology for asynchronous NoC generation and simulation. Our evaluation methodology and setup is given in Section VI, with the results discussed in Section VII. We conclude in Section VIII.

II. Related Work

The COSI framework [5] generates an application-specific NoC and floorplan, taking as input such constraints as core areas and average bandwidth between cores. While it is extensible with new algorithms and components, it does not consider asynchronous network components and, as future work, cites the need for integrating traffic burstiness. For the Xpipes library, a heuristic search determines the topology and router configuration [6]. It uses floorplan information, router energy models, and core communication requirements. The results indicate a significantly reduced power and hop-count versus the best mesh topologies for a variety of SoC designs. It is part of a complete workflow to automatically synthesize a NoC down to chip layout [7]. A linear programming based method is presented in [3]. For the QNoC routers, application-specific optimization is discussed in [8], but it focuses on mapping logical resources of a mesh-style topology rather than physical concerns.

Previous research on asynchronous interconnects is rich, but these designs are either hand-designed for a particular application, or general in design but possibly having over-provisioned resources for a power-constrained SoC. All but one of these existing routers use quasi delay-insensitive protocols between routers, rather than bundled-data. Fulcrum Microsystems created a large asynchronous crossbar to interconnect cores of a SoC [9]. The commercial startup Silistix, based on earlier academic research [10], sells EDA software and circuits that provide an customized asynchronous NoC, but has no published methods for the optimization process. The MANGO router [11] provides both best-effort and guaranteed-service traffic. FAUST [12] is a platform and fabricated chip used in 4G telephony development, and uses an asynchronous mesh-based NoC [13]. The QNoC group has developed an asynchronous router that provides multiple service levels and dynamically allocated virtual channels per level [14]. A mesh-of-trees network was constructed from simple, bundled-data routers for a CMP [15]. A comparison between the asynchronous network ANOC, and the mesochronous clocked network DSPIN, was performed in [1]. For both designs, a physical layout and functional traffic simulation was done for analysis. While DSPIN had 33% less area and 33% higher bandwidth than ANOC, ANOC had shorter packet latency and at least 37% lower power consumption. DSPIN was also compared against its asynchronous analog, ASPIN [16]. Average power, latency, and saturation threshold are superior in ASPIN with similar die area.

Traffic modeling for NoCs is one of the major outstanding problems in the field [17]. The *b*-model [18] provides a simple method to produce and analyze the burstiness of self-similar traffic with a single value. The *b*-model has been adapted to study burstiness effects in the Nostrum NoC [19]. Evidence of traffic self-similarity and burstiness in MPEG-2 video applications has been shown [20]. Several analytic models of network performance have been developed for NoC design. A model has been developed to capture spatial and temporal characteristics of traffic for regular, homogeneous NoCs [21]. A generalized analytic router model was developed in [22]. It provides detailed statistics during expected traffic, and is applicable to heterogeneous, irregular networks, but relies on the Poisson arrival process and a synchronously-clocked router.

III. Synchronous Network Generation

The baseline network used for comparison purposes is generated by a research tool called COSI 2.0, a source-code release that incorporates much of the functionality of COSI-NoC (v.1.2) [5]. COSI takes as input a SoC design abstraction consisting of core dimensions or area, and a set of communication constraints between those cores, which are called *flows*. This is a more generalized concept than the CTG mentioned in Section I, and COSI can consider temporal properties between flows, such as mutual exclusion. Given these flows, its optimization algorithms try to find the network and floorplan that meets the constraints while minimizing power based on router and wire models. As output, COSI produces a floorplan, topology, and a SystemC-based simulator. Included with the software release are algorithms for generating a mesh and a min-cut partitioning method (hierarchical star) similar to that of [6]. We modified COSI to incorporate the Orion 2.0 router and wire models [23], and also made a number of other changes to COSI to improve its operation and result reporting.

In order to explore the performance characteristics of the network, we moved away from the Poisson traffic models commonly used for evaluations and instead use a model more representative of application traffic. We implemented the *b*-

model traffic generator [18], suggested as a key feature in future NoC benchmark sets [24]. The SystemC simulator produced by COSI was modified to use this bursty traffic generator.

Our model is parameterizable with the following inputs:

- Source and destination cores.
- A b-value in the range $[0.5, 1.0)$ indicating burstiness.
- Simulation duration.
- Average bandwidth, i.e. desired total traffic volume.
- The smallest time-resolution of the burstiness.
- Number of packets per message.

Self-similar traffic, down to the time resolution, is generated recursively with an algorithm closely following the original [18]. However, there are a number of interesting details to note. The b-model determines the total volume of data to send in each *window* determined by the specified time resolution. Within a window, a message is probabilistically sent each cycle such that over the time window the proper amount of data is sent. An entire message consisting of multiple packets is sent at once to emulate application-level data needs. It may be the case that the desired volume of traffic per window exceeds the capacity of the link or output buffer, or the previous window has not finished sending its data yet. In these cases the packets are queued up in an "infinite" buffer. Therefore, the model's output is the ideal, desired data transmissions, but the actual achieved data is subject to network limitations as expected. This design uses a SystemC transaction level model (TLM) for its interface, and thus it is portable and relatively easy to connect to other tools' outputs, as we did here with COSI.

IV. Asynchronous Router Design

A. Overview

This asynchronous router is designed for efficiency and simplicity. Each switch directs a flit to one of two output ports. With bi-directional channels, this results in a three-ported "T" router. The packet format consists of a single flit containing source-routing bits in parallel, on separate wires, with the data bits. The packet is switched through a simple demultiplexer controlled by the most-significant routing bit. The bits are simply rotated, or swizzled, for the output packet. The number of required routing bits is determined by the maximum hop count of a network generated for a specific SoC design. The width of each flit must be determined based on required throughput or power and area constraints. This format has the overhead of requiring routing bits with every flit.

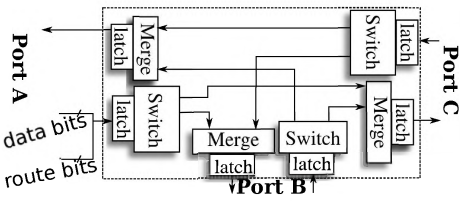


Figure 2: Architecture of a 3-port asynchronous router.

The router is implemented with three components: a switch module, merge module, and a buffer. The switch module steers data on an incoming channel to one of the other two outgoing channels. The merge module arbitrates between two input channels to an output channel, granting access to the first-to-arrive request signal. This effectively alternates between the two input channels, assuming each provides the next packet within an output channel's cycle-time. A router is composed of three switch modules and three merge modules, as shown in Figure 2. Each switch and merge module has one set of latches providing 1-flit buffers on each input and output port.

B. Router Circuit Design

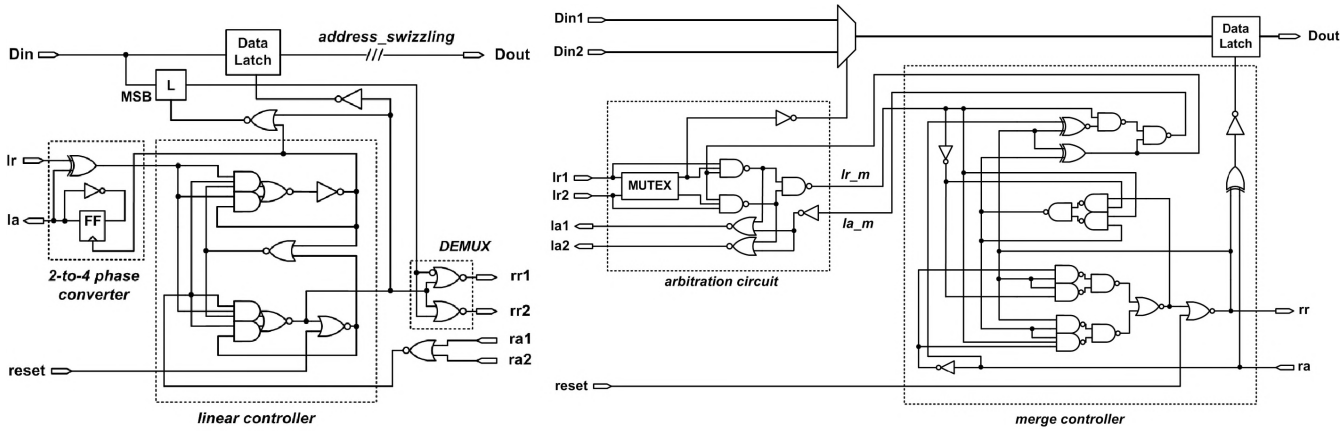
Asynchronous protocols normally fall into two categories: quasi delay-insensitive (QDI) and bundled-data (BD). Generally, QDI is more robust to variations while BD allows simpler circuits. BD has a lower wire count compared to QDI's common encodings (e.g. 1-of-4 and dual-rail). This is potentially more energy-efficient due to reduced wire repeater leakage, especially with wide links [25]. The choice of 4-phase or 2-phase protocol impacts performance and circuit complexity. The throughput across long links is limited by wire latency, thus a 2-phase protocol achieves almost twice the throughput as a 4-phase protocol. However, a 4-phase, level-sensitive protocol typically allows more simple circuits.

With this in mind, we designed the router to internally operate using a BD 4-phase protocol since it directly works with a level-sensitive 4-phase MUTEX element [26] used for arbitrating the shared output channels. We employ a BD 2-phase protocol on the channels between routers.

The design of the router's switch module is shown in Figure 3a. A 2-to-4 phase converter is implemented on the input control channel (signals *lr* and *la*). This handshakes with a BD 4-phase burst-mode asynchronous controller to pipeline the data. The linear controller has the same specification and timing assumptions as the one used in [27]. The output request is steered to one of two channels (*rr1* or *rr2*) based on the most significant route bit with a DEMUX. The route-bits are rotated and passed to the merge module of the router. The routing logic occurs concurrently with the handshake.

The merge module is composed of the arbitration circuit and merge controller shown in Figure 3b. It contains the arbiter that serializes requests to the shared output channel. The output of the arbiter controls a MUX that selects which input data to store in the output latch. Each arbiter transaction requests a data transfer via the 4-phase handshake signal *lr_m*. This request passes through the merge controller to generate the 2-phase network link handshake on signals *rr* and *ra*, as well as store the data in a pipeline latch.

All of the circuits were designed with the static, regular V_{th}, Artisan cell library on IBM's 65nm 10sf process except the MUTEX element in the merge module. We designed and characterized a separate library cell for the MUTEX element through manual layout and HSPICE simulation. This asynchronous circuit design process used a clocked CAD flow in a methodology similar to [28]. We synthesized our



(a) switch (b) merge

Figure 3: Schematics of Switch and Merge modules.

asynchronous controllers by hand or using Petrify, synthesized the full asynchronous structural router design including data path with Synopsys Design Compiler, and physically placed and routed with SOC Encounter. Functionality and performance were validated in the design with ModelSim using back annotated pre- and post-layout delays. Asynchronous circuits were verified by Analyze [29] and using static timing analysis.

C. Evaluation

We have constructed a number of routers with varying flit widths, but for this paper use one with 32-bits of data and 8-bits for routing. The resulting area is 2740 μm^2, dynamic energy/flit is 1.56 pJ, and leakage power is 0.009 mW.

The area is dominated by data storage latches and the data MUXes used in the merge modules. The controllers (linear controllers in switch modules and merge controllers in merge modules) make a very small contribution to the total area. Dynamic energy is consumed when one data word passes a router from an input port to an output port. Energy is measured using HSPICE simulations with the spice netlist generated from the design using parasitic extraction from Mentor Graphics Calibre PEX. The same simulation was used in both HSPICE and ModelSim. The HSPICE control file was generated by converting a vcd file generated from the ModelSim simulation. This allowed us to more easily validate switching activity on the data and control paths. A 50% data switching activity factor was applied to the data bits for our power simulations.

The maximum throughput of the router is 2.38 Gflits/s. This was measured by inserting data into the input ports at maximum rate and allowing the output port to communicate with another router with no wire delay.

We define the backward latency of our routers as the delay from a request on an incoming channel to the acknowledgment on that channel, completing the handshake of the two-phase protocol. Fast backward latency is desirable because it frees the previous router's output port for another transaction. We define forward latency as the delay from a request on an incoming channel of a router to the associated request on an output channel assuming no contention or stalling in the arbitration circuit. This is determined by the delay to buffer the data, arbitrate control, and switch to the outbound channel. Our router design has 250 ps backward latency and 460 ps forward latency.

Our router's low power and area are due to its simple architecture and the use of latches, rather than flip-flops, for the storage elements. Latches are about half the size and use less power than flip-flops. Since much of the area and power of many router architectures derives from memory elements, this advantage makes a significant difference. Furthermore, the simplicity of the control circuits also contributes to high throughput. This router employs a bundled data protocol rather than delay insensitive codes which results in fewer wires per channel and efficient use of standard cell libraries. However, the cost to this is that the circuit timing must be carefully specified and controlled, similar to clocked design, to ensure correct operation.

V. Asynchronous Network Generation

We built a tool, *ANetGen*, that has goals similar to COSI's, but operates with our router model and its asynchronous considerations. ANetGen takes an input format that defines the CTG edges and expected traffic bandwidth, as well as the core dimensions. The core floorplan is specified prior to ANetGen, which then determines physical placement of the routers and their logical topology. The objective function is to minimize wirelength and hop counts for high traffic paths. It does this with a combination of simulated annealing (SA) and force-directed movement techniques.

A. Topology and Placement

Asynchronous circuits have unique properties that can be leveraged to optimize the network. Specifically, the physical path length between endpoints directly affects packet latency, not just the number of routers and pipeline buffers a packet must travel through, assuming an uncongested path. This is in contrast to a synchronous system, where each network element constitutes at least one required clock cycle. Also, link energy

usage can be significant [30] and will grow, relatively, with shrinking process technology.

With this in mind, the physical placement of routers needs to be determined such that wirelength is minimized, especially on highly trafficked paths. For these experiments, we assume soft IP (intellectual property) blocks which have cells placed and routed by the SoC developer, rather than a single hard macro block. This enables us to consider more options for router locations. In an actual design flow, the router placement our tool generates will provide input to the hierarchical placer, or floorplacer [31] that will legalize the placement of cells and macros composing each core.

The problem of finding the optimal tree topology is similar to the NP-hard quadratic assignment problem of mapping cores to a mesh topology [32]. For this, we utilized a simulated annealing method. The fitness to be minimized is based on a topology's router hop-count and wirelength, each weighted by the volume of traffic expected over the path. In the current tool implementation we limit the topology to a tree, which has a minimal number of three-port routers. We save a detailed analysis and comparison with other topologies to future work, but this method produces good results, as seen in Section VII.

Within the SA process, topology choices are explored by perturbing the topology and re-placing routers. We used a method extended from [33] that uses force-directed movement to provide router locations and link lengths to the SA process. Force vectors are applied to routers that are proportional to: (a) bandwidth requirements and (b) physical distances between the router and its attached core or router. The process is iterative, where a router moves a distance proportional to the sum of its force vectors. This movement will reduce wire lengths of paths that carry high traffic.

B. Simulator

We chose to build an asynchronous network simulator using the SystemC library. The following modules were developed: an arbiter, an *inport* to the router, an *outport* from the router, and input and output port FIFOs. The SystemC Transaction Level Modeling (TLM) library is used for inter-router links and traffic generation. We chose this method to allow easier extensibility of the channels if needed, and TLM provides a convenient way to model link and protocol delays.

The traffic generator and router ports use a `simple socket` to receive a `generic payload` transaction object that contains packet and routing information. When a TLM object is received by the *inport*'s socket, a `wait` is performed to model the wire delay. This delay is calculated from an interpolation of HSPICE simulations of various wirelengths in IBM's 65nm technology. The wire energy per transfer is calculated using the Orion 2.0 model. The router waits an additional time period to model forward logic delay. The flit is written to the FIFO, which triggers the arbiter. Another `wait` models the acknowledgment delay to the sender.

Within the arbiter, a `doSwitching` SC_METHOD is called whenever a packet is received by an input FIFO or acknowledged by an output FIFO. The arbitration mechanism is that described in Section IV. At each switching operation, the appropriate energy is logged. This energy was measured from transistor-level router simulations.

Each *outport* operates in its own thread, waiting for a packet to be passed to it by the arbiter, or for a TLM response indicating that the channel is free. When there is data in the FIFO and the channel is free, it sends a new TLM `generic payload`. The *outport* also records wire energy of the transmitting link.

VI. Evaluation Methodology

The evaluation of all network solutions was done with the SystemC simulators generated by the tools. In this section, we present the benchmarks and simulation parameters.

A. SoC Designs and NoC Generation

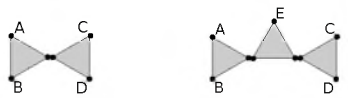
We used two SoC design abstractions of the CTG format described in Section I for our evaluations. One is titled ADSTB and is from the public COSI 2.0 distribution. The other is an MPEG4 decoder originally described by [34] and used in several other NoC research projects. Bandwidth requirements were modified from those originally provided, and are shown in Figure 1 for MPEG4 and Table I for ADSTB. The die areas after router placement for the ADSTB and MPEG4 designs were 35.7 mm^2 and 78.7 mm^2, respectively. These floorplans were from the COSI tool's output.

TABLE I: Average bandwidths for the ADSTB design.

Sender	Receiver	MBytes/s	Sender	Receiver	MBytes/s
CPU	AudioDec	1	CPU	DDR	3
CPU	Demux	1	CPU	MPEG2	1
DDR	CPU	3	DDR	HDTVEnc	314
DDR	MPEG2	593	Dem1	Demux	31
Dem2	Demux	31	Demux	AudioDec	5
Demux	MPEG2	7	HDTVEnc	DDR	148
MPEG2	DDR	424			

We generated a network for each design using the COSI and ANetGen tools. We also manually created an asynchronous network for the ADSTB design that is based on the topology of the COSI solution. For each radix-4 and radix-5 router, we manually replaced it with a construction of our radix-3 asynchronous routers, shown in Figure 4. The paths which carry the most traffic were mapped to ports with the least number of routers between them, such as ports A and B. This construction is not a true radix-N switch, as it can have internal contention (e.g. $A \rightarrow C$ contends with $B \rightarrow D$).

We configured COSI to generate a hierarchical star network with $N/3+1$ partitions (N is number of cores), chosen based on empirical experimentation for low energy. The floorplanner



(a) Radix-4 (b) Radix-5

Figure 4: Asynchronous router constructions replacing those of $radix > 3$. External ports are labeled with letters.

was constrained to a square aspect ratio outline. The input to ANetGen was the same floorplan and communication properties as COSI.

B. Simulation Parameters

We instrumented the SystemC router and wire models from COSI and ANetGen to record energy usage, packet latency, and message latency over the course of a simulation. Orion 2.0 is used for the wire energy model in both frameworks, and also for the synchronous router leakage power and switching energy models. Energy for the asynchronous routers comes from circuit simulation described in Section IV. The link model assumes 50% of the wires switch per flit transfer. This is a worst case model because real data will have a lower fraction of changing bits. Additionally, the asynchronous router's source-route wires will change less than this as subsequent flits often carry similar routing paths. Thus, the overhead of these additional routing wires is likely less than what is represented in the results.

We chose parameters for the Orion router model to be near as possible to our asynchronous configuration in both energy and performance. These are shown in Table II. Clock tree power estimation was excluded from these models.

TABLE II: Orion 2.0 Model Parameters.

Router Freq.	2 GHz	Router I/O buff's	2 / 1 flit
Tech. Library	65 nm NVT	Crossbar	Multitree
Voltage	1.0 v	Flit width	32 bits

VII. Results

In this section we present results that show the asynchronous networks provided lower message latency and used less power than the synchronous networks.

Recall that a message is composed of a number of packets, and is typically managed at the transport layer. Message latency is defined as the time the first packet of the message leaves the sending core's output buffer and enters the network to the time the tail packet leaves the network and enters the destination core. The following results were generated with a message size of 256 bytes, not counting flits carrying address information. Simulations were run at three burstiness b-values $\{0.5, 0.65, 0.8\}$. We assume that packets are not dropped, and that the destination cores do not stall, blocking its input port.

A. Message Latency Distribution

Histograms of message latency are shown in Figure 5 for the ADSTB design, and a summary of both is presented in Table III. An increase in latency as traffic burstiness rises shows that traffic paths contend for switch and link resources for longer periods of time. At 0.5 burstiness, all networks operate with low latency of 150-190 ns for nearly all traffic. At 0.8 burstiness, the asynchronous networks have more messages arriving in under 200ns, and a lower "re-peak" on the right side of the chart.

TABLE III: Observed message latencies (ns); absolute maximum and latency bound of 99%.

	Network	Burstiness		
99% less than		**0.5**	**0.65**	**0.8**
ADSTB	sync.	158	231	531
	manual async	188	262	274
	ANetGen	192	291	304
MPEG4	sync.	838	1395	1903
	ANetGen	275	431	697
Maximum				
ADSTB	sync.	1130	51077	126480
	manual async	510	580	914
	ANetGen	510	762	912
MPEG4	sync.	11722	56041	158264
	ANetGen	704	2520	5288

B. Per-path Message Latency

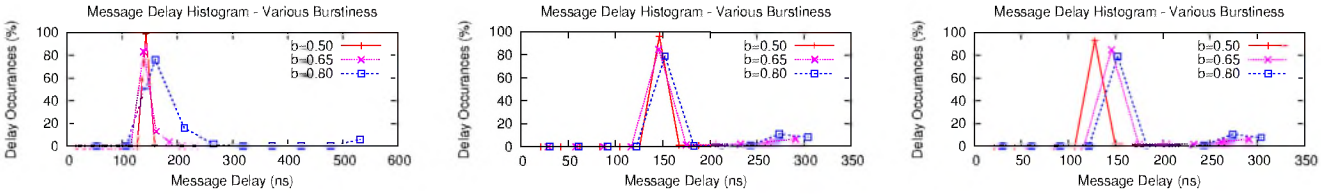
An understanding of latency and congestion within the network cannot be fully understood by the overall delay alone. Due to the heterogeneity and diverse path properties in an application-specific SoC, there is benefit to analyzing each path through the network separately.

For each path, or pair of communicating cores, Figure 6 shows the maximum latency for the ADSTB design and the median latency for the MPEG4 design. With few exceptions, the asynchronous network provides lower latency, at both low and high burstiness values. This is a combination of a number of factors. COSI's synchronous routers operate with wormhole switching in which a blocked header flit stalls up to two trailing flits that in turn block other packets in other routers. Second, the asynchronous network can take advantage of short wires between routers and not delay a packet an entire cycle. Lastly, the COSI-based router design has the overhead of an extra flit carrying address information.

Despite the large variation in some paths, other paths show little difference between change in burstiness or between median and max delay. This is due to the network topology, where some paths have fewer hops and a lower chance of congestion, and others must traverse more routers. Both COSI and ANetGen map paths carrying more traffic such that they have fewer router hops.

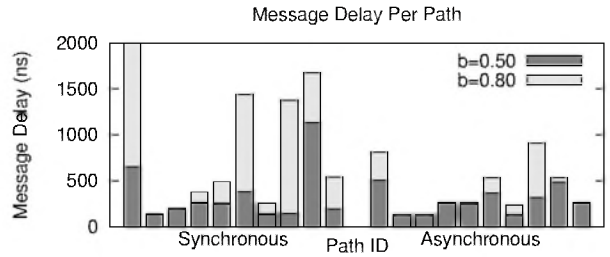
C. Output Buffer Delay

Another metric of measuring the network performance is the output buffer delay, which is from the time the traffic generator places a packet in the output buffer to the time the packet enters the network. The buffer entry time is set for each packet by the traffic generator when it pushes an entire message to the buffer at once. Therefore, the last packet of a 64-packet message would have a minimum delay of 64 sender-cycles. The traffic generator operates detached from the network flow control so an infinite buffer is needed to accept its traffic at any time. The network then empties that buffer as quickly as possible. As burstiness increases, the additional delay comes not only from contention within the network, but also from the local traffic generator's attempt to send more data in a shorter time period. This grows the buffer more rapidly, increasing delay, even if the network was uncongested. From the results in

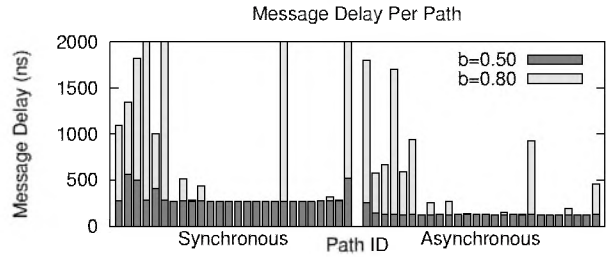


(a) COSI-generated synchronous (b) Manual asynchronous (c) ANetGen asynchronous

Figure 5: Histograms of message latency for the ADSTB design.



(a) Maximum latency for ADSTB



(b) Median latency for MPEG4

Figure 6: Observed latencies per path for 256-byte messages. Results are shown for the synchronous and asynchronous network, and traffic of two burstiness amounts.

Table IV, we see that the asynchronous networks consistently have a lower delay for both median and maximum values.

D. Instantaneous Bandwidth

A measure of network performance related to message latency is the instantaneous bandwidth (IB) available to a path at any given time. This is in contrast to the average bandwidth that an application produces over a long period. We define an IB requirement for a source-destination path by a pair of values: $\{V, T\}$, where V is in bytes and T is in seconds. This might be used in validating an interconnect of, say, a real-time speech recognition SoC, where 18 MB must be processed in 0.1 s [35].

For example, the maximum synchronous network latency seen in simulation between the *upsamp* and *sdram* cores of the MPEG4 was 2525 ns. Suppose this path had an IB requirement of $\{256\,bytes, 1000\,ns\}$ (equating to 256 MBytes/s). This network would be a poor choice because the application would occasionally not receive proper communication throughput, despite the fact that the network did support its average bandwidth.

TABLE IV: Source output buffer packet delay (ns).

	Network	Burstiness		
Median		**0.5**	**0.65**	**0.8**
ADSTB	sync.	96	730	390065
	manual async.	90	508	320018
	ANetGen	90	500	319818
MPEG4	sync.	274	170336	1.1e6
	ANetGen	84	250	171496
Maximum				
ADSTB	sync.	1274	261112	1.3e6
	manual async.	952	215908	1.2e6
	ANetGen	912	231242	1.2e6
MPEG4	sync.	11683	1.2e6	2.99e6
	ANetGen	1036	171358	1.1e6

E. Power Consumption and Area

We present the power consumption in Table V, broken down into the following areas: dynamic power of routers, leakage power of routers, dynamic power of wires, and leakage power of wires. These measurements do not include the power of clock distribution, and assume clock gating at the router. The wire power includes that from drivers and repeaters (large inverters).

TABLE V: Power consumption (mW) of routers and wires.

	rtr dyn	rtr leak	wire dyn	wire leak	TOTAL
ADSTB sync	5.5	5	7.86	4.6	23
manual async	0.95	0.072	12	8	21
ANetGen	0.95	0.054	6.3	4.5	11
MPEG4 sync	12.3	15.7	28	15	71
ANetGen	2.4	0.09	20.5	16.7	40

In both cases, the dynamic power of the asynchronous routers is reduced to about one-fifth the power of the synchronous routers. The leakage power of the asynchronous routers is negligible. The manual asynchronous network for the ADSTB design has a noticeable increase in wire power. One reason is the additional links needed to form the cluster of 3-port routers in place of a higher-radix router. Second, the routing bits are on separate wires, rather than an address on the head flit. Third, our routers and tools use bi-directional ports, with links instantiated in both directions. COSI, meanwhile, considers uni-directional router ports, and may produce a solution with fewer links.

Overall, the asynchronous networks use less power than the synchronous networks. The majority of savings comes from significantly lower router power, both dynamic and leakage. These results also point to the need for wire resources to be careful utilized, especially with energy-efficient routers.

The ANetGen networks have an area advantage over the synchronous ones as well. Router areas are $15630\,\mu m^2$ (ANetGen) vs. $99704\,\mu m^2$ (COSI) for ADSTB, and $26050\,\mu m^2$ vs. $138822\,\mu m^2$ for MPEG4.

VIII. Conclusion

In this paper we provided an examination of performance and energy of synchronous and asynchronous NoCs that are customized for a number of SoC designs. The asynchronous network formed by our tool ANetGen and our energy-efficient routers only consumed half as much power as the synchronous case. Wires consumed the largest fraction due to repeater energy. Our asynchronous network also had lower latency, significantly so for bursty traffic, for 256-byte messages. For the ADSTB design, ANetGen yielded a lower-energy solution and slightly lower latency than a manually-designed network based on the synchronous topology. The evaluation suggests that the common abstraction of SoC requirements using only average bandwidth may not be sufficient. The addition of a single-valued burstiness to the tuple representing a network flow's properties should be considered in other NoC evaluations. In future work, we will refine ANetGen to consider a wider range of topologies and more closely integrate it with the floorplanning tool.

Acknowledgments

This research is supported by the National Science Foundation under grant CCF-0702539 and Semiconductor Research Corporation under task 1817.001. We would like to thank ARM and IBM for providing the 65nm library cells and process technology.

References

[1] I. M. Panades, F. Clermidy, P. Vivet, and A. Greiner, "Physical implementation of the dspin network-on-chip in the faust architecture," in *Proc. Int'l Symp. on Networks-on-Chips*, 2008, pp. 139–148.

[2] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar, "A 5-ghz mesh interconnect for a teraflops processor," *IEEE Micro*, vol. 27, no. 5, pp. 51–61, 2007.

[3] K. Srinivasan, K. S. Chatha, and G. Konjevod, "Linear-programming-based techniques for synthesis of network-on-chip architectures," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 14, no. 4, 2006.

[4] B. R. Quinton, M. R. Greenstreet, and S. J. E. Wilton, "Practical asynchronous interconnect network design," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 16, no. 5, pp. 579–588, 2008.

[5] A. Pinto, L. P. Carloni, and A. L. S. Vincentelli, "A methodology for constraint-driven synthesis of on-chip communications," *IEEE Transactions on Computer Aided Design*, vol. 28, no. 3, pp. 364–377, 2009.

[6] S. Murali, P. Meloni, F. Angiolini, D. Atienza, S. Carta, L. Benini, G. D. Micheli, and L. Raffo, "Designing application-specific networks on chips with floorplan information," in *Proc. Int'l Conf on Computer-Aided Design*, 2006, pp. 355–362.

[7] D. Atienza, F. Angiolini, S. Murali, A. Pullini, L. Benini, and G. De Micheli, "Network-On-Chip Design and Synthesis Outlook," *Integration-The VLSI Journal*, vol. 41, no. 2, Feb. 2008.

[8] Z. Guz, I. Walter, E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "Network delays and link capacities in application-specific wormhole nocs," *VLSI Design*, vol. 2007, 2007.

[9] A. Lines, "Asynchronous interconnect for synchronous soc design," *IEEE Micro*, vol. 24, no. 1, pp. 32–41, 2004.

[10] W. J. Bainbridge and S. B. Furber, "CHAIN: A Delay Insensitive CHip Area INterconnect," *IEEE Micro special issue on Design and Test of System on Chip*, vol. 142, No.4., pp. 16–23, Sept. 2002.

[11] T. Bjerregaard and J. Sparsø, "A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip," in *Proc. Design, Automation, and Test in Europe*, 2005, pp. 1226–1231.

[12] D. Lattard, E. Beigne, C. Bernard, C. Bour, F. Clermidy, Y. Durand, J. Durupt, D. Varreau, P. Vivet, P. Penard, A. Bouttier, and F. Berens, "A telecom baseband circuit based on an asynchronous network-on-chip," *Solid-State Circuits Conference, Digest of Technical Papers*, Feb. 2007.

[13] E. Beigne, F. Clermidy, P. Vivet, A. Clouard, and M. Renaudin, "An asynchronous noc architecture providing low latency service and its multi-level design framework," in *Proc. Int'l Symposium on Asynchronous Circuits and Systems*, 2005, pp. 54–63.

[14] R. R. Dobkin, R. Ginosar, and A. Kolodny, "Qnoc asynchronous router," *Integr. VLSI J.*, vol. 42, no. 2, pp. 103–115, 2009.

[15] M. N. Horak, "A high-throughput, low-power asynchronous mesh-of-trees interconnection network for the explicit multi-threading (xmt) parallel architecture," Master's thesis, Univ. of Maryland, August 2008.

[16] A. Sheibanyrad, I. M. Panades, and A. Greiner, "Systematic comparison between the asynchronous and the multi-synchronous implementations of a network on chip architecture," in *Proc. Design, Automation, and Test in Europe*, 2007, pp. 1090–1095.

[17] R. Marculescu, U. Ogras, L.-S. Peh, N. Jerger, and Y. Hoskote, "Outstanding research problems in noc design: System, microarchitecture, and circuit perspectives," *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, vol. 28, no. 1, pp. 3–21, 2009.

[18] M. Wang, T. Madhyastha, N. H. Chan, S. Papadimitriou, and C. Faloutsos, "Data mining meets performance evaluation: fast algorithms for modeling bursty traffic," in *Proc. International Conference on Data Engineering*, 2002.

[19] R. Thid, I. Sander, and A. Jantsch, "Flexible bus and noc performance analysis with configurable synthetic workloads," in *Proc. EUROMICRO Conf on Digital System Design*, 2006, pp. 681–688.

[20] G. V. Varatkar and R. Marculescu, "On-chip traffic modeling and synthesis for mpeg-2 video applications," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 12, no. 1, pp. 108–119, 2004.

[21] V. Soteriou, H. Wang, and L.-S. Peh, "A statistical traffic model for on-chip interconnection networks," in *Proc. Int'l Symp. on Modeling, Analysis, and Simulation*, 2006, pp. 104–116.

[22] U. Y. Ogras and R. Marculescu, "Analytical router modeling for networks-on-chip performance analysis," in *Proc. of Design, Automation and Test in Europe*, 2007, pp. 1096–1101.

[23] A. Kahng, B. Li, L.-S. Peh, and K. Samadi, "Orion 2.0: A fast and accurate noc power and area model for early-stage design space exploration," in *DATE*, April 2009, pp. 423–428.

[24] Z. Lu, A. Jantsch, E. Salminen, and C. Grecu, "Network-on-chip benchmarking specification part 2: Micro-benchmark specification," *Technical Report, OCP International Partnership Association, Inc.*, 2008.

[25] K. S. Stevens, P. Golani, and P. A. Beerel, "Energy and Performance Models for Synchronous and Asynchronous Communication," *IEEE Tran. on VLSI Systems*, March 2010.

[26] C. Seitz, *System timing*. Addison-Wesley, 1980, ch. 7, in C.A. Mead and L.A. Conway, editors, Introduction to VLSI Systems.

[27] K. S. Stevens, R. Ginosar, and S. Rotem, "Relative timing," *IEEE Trans. on VLSI Systems*, vol. 11, no. 1, pp. 129–140, 2003.

[28] K. S. Stevens, Y. Xu, and V. Vij, "Characterization of asynchronous templates for integration into clocked cad flows," in *Int'l Symp. on Asynchronous Circuits and Systems*, May 2009, pp. 151–161.

[29] K. S. Stevens, "Practical verification and synthesis of low latency asynchronous systems," Ph.D. dissertation, Univ. of Calgary, Sept. 1994.

[30] A. Pullini, F. Angiolini, P. Meloni, D. Atienza, S. Murali, L. Raffo, G. D. Micheli, and L. Benini, "NoC design and implementation in 65nm technology," in *Proc. Int'l Symp. on Networks-on-Chip*, May 2007.

[31] J. A. Roy, D. A. Papa, S. N. Adya, H. H. Chan, A. N. Ng, J. F. Lu, and I. L. Markov, "Capo: robust and scalable open-source min-cut floorplacer," in *Proc. Int'l Symp. on Physical Design*, 2005, pp. 224–226.

[32] J. Hu and R. Marculescu, "Energy-aware mapping for tile-based noc architectures under performance constraints," in *Proc. Asia and South Pacific Design Automation Conference*, 2003, pp. 233–239.

[33] D. Gebhardt and K. S. Stevens, "Elastic flow in an application specific network-on-chip," *Electron. Notes Theor. Comput. Sci.*, vol. 200, no. 1, pp. 3–15, 2008, Proc. Int'l Workshop on Formal Methods for GALS.

[34] E. B. V. D. Tol and E. G. T. Jaspers, "Mapping of mpeg-4 decoding on a flexible architecture platform," in *Media Processors*, 2002, pp. 1–13.

[35] B. Mathew, A. Davis, and Z. Fang, "A low-power accelerator for the sphinx 3 speech recognition system," in *Proc. on Compilers, Architecture and Synthesis for Embedded Systems*. ACM, 2003, pp. 210–219.